Tutoriel Android

Pour démarrer la programmation d'applications Android



Remarque préalable

Comme il est impossible de traiter tous les aspects d'Android, seulement quelques uns seront présentés.

De plus pour des raisons pédagogiques, ils seront présentés dans un ordre de complexité croissante et de proximité de centres d'intérêts (c'est à dire au fil de l'eau, comme un tutoriel).

Plan

- Quelques mots de présentation
- Installation du SDK (outils de programmation)
- Première application et premiers concepts
- Application Liste d'items
- Application Dessins

Présentation

- Android est né en 2007
- C'est :
  - Un système d'exploitation pour tablettes et smartphones reposant sur un système Linux
  - Une machine virtuelle Java (dalvik) établie sur d'autres bases que la machine Java standard
  - Une API de programmation en Java :
    - Énorme quantité de fonctionnalités
    - Outils de développement pour Eclipse

Programmation de logiciels

- Cycle de travail :
  - Développement avec Eclipse en Java
  - Cross-compilation
  - Installation sur la cible (liaison USB)
  - Débogage et tests du logiciel
- Outils :
  - Plugin transparent et très bien fait dans Eclipse
  - SDK avec des outils de gestion (adb notamment)

SDK, AVD, ADB et ADT

SDK = outils et fichiers pour programmer
AVD = tablettes virtuelles
ADB = logiciel de communication
ADT = plugin pour Eclipse

Le SDK Android

Il contiendra, après téléchargement(s) :

- la doc en ligne
- les fichiers inclus, les librairies liées...
- le compilateur et autres outils
- des outils de communication genre telnet et ftp
- un émulateur de tablette Android virtuelle

- 1) Le prendre sur :

http://developer.android.com/sdk/index.html

  - Demander le SDK correspondant au système

Installation du SDK

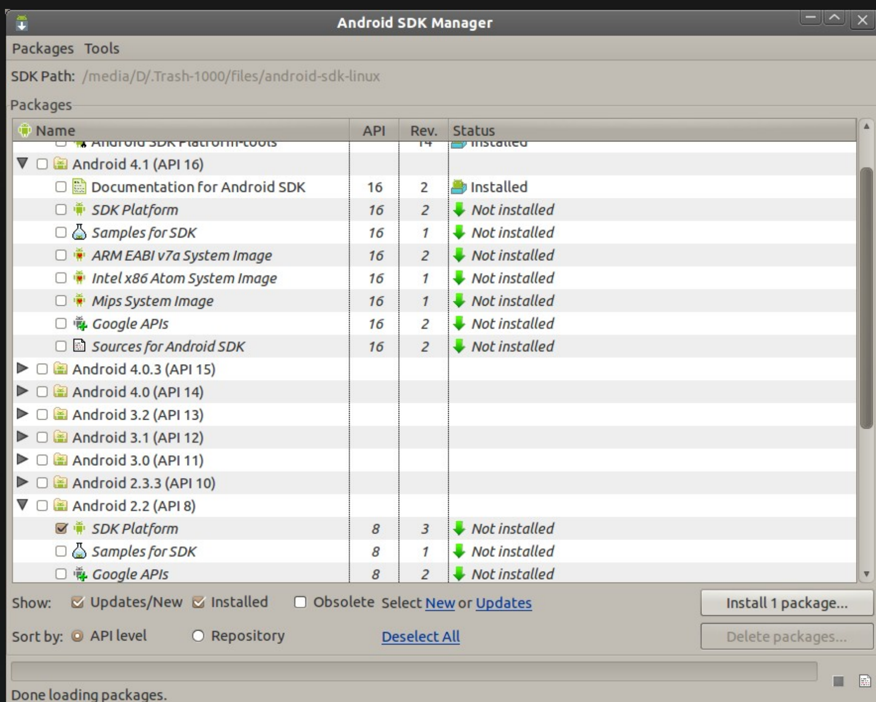
- Prérequis : avoir installé Java
- 2) Extraire les fichiers de l'archive ou lancer l'exe

    => android-sdk-linux ou android-sdk-windows

- 3) Lancer tools/android ou tools\android.bat

    => ouvre une fenêtre permettant de sélectionner les composants du SDK qu'on souhaite installer et/ou mettre à jour

Android SDK Manager



- 4) Dans Tools, vérifier que tout est « Installed »
- Dans Android 4.1 :
  - Décocher tout (la doc reste Installed)
- Dans Android 2.1 ou 2.2 :
  - Cocher uniqt SDK Pltform
- 5) Cliquer **Install 1 package** (pas de delete)

Quelques mots sur les versions

- Installer la version la plus basse commune aux tablettes ciblées par vos logiciels
  - Une appli 4.1 ne tournera pas sur une tablette 2.3
- Obsolescence voulue explicitement, pourtant, en octobre 2012 :

4.1.x Jelly Bean	16	07/2012	2%
4.0.x Ice Cream Sandwich	18,19	10/2011	23%
3.x.x Honeycomb	11-13	02/2011	2%
2.3.x Gingerbread	9,10	12/2010	56%
2.2.x Froyo	8	05/2010	13%
2.1 Eclair	5-7	10/2009	3%
1.6 Donut	4	09/2009	0,4%
1.5 Cupcake	3	04/2009	0,1%

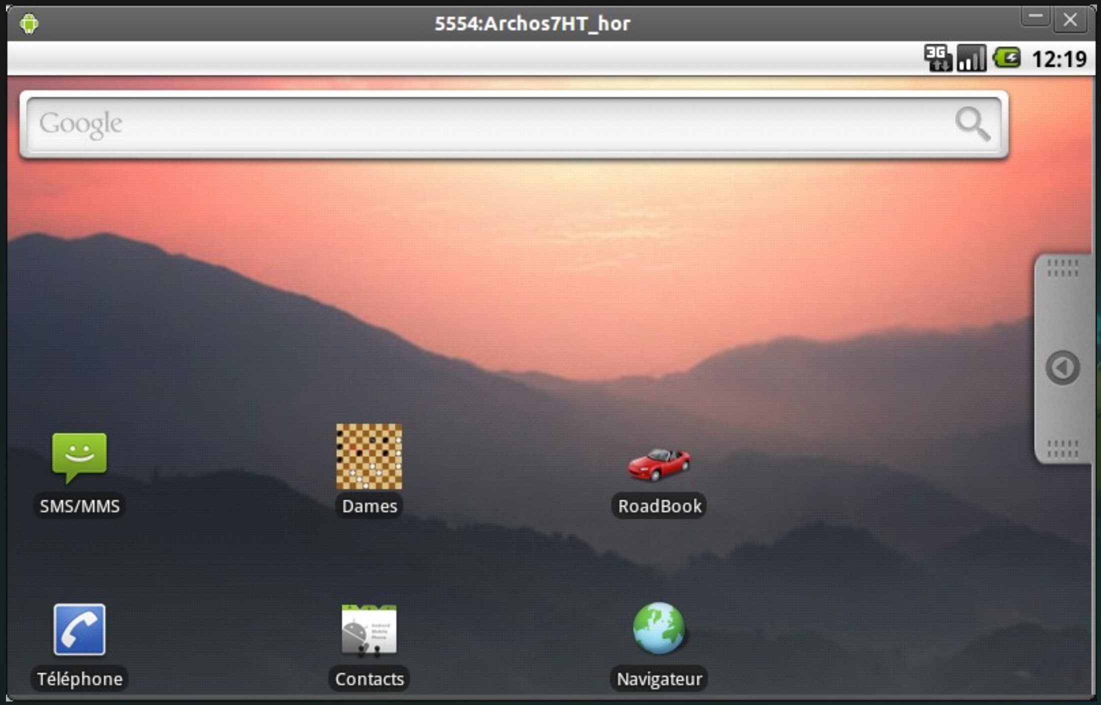
Android SDK Manager (suite)

- Ce gestionnaire rajoute de nombreux éléments dans votre dossier SDK et permet de les maintenir à jour
- Critiques :
  - La doc qu'on installe est uniquement la plus récente, elle ne correspond pas à la plate-forme voulue => nombreux avertissements d’obsolescence
- Rajouter platform-tools et tools dans le PATH

  → commandes (adb, android) disponibles partout

Émulateur de tablettes virtuelles

- Le SDK android contient un **émulateur de tablette** permettant de faire tous les tests sans lessiver la mémoire flash de la tablette



Créer une tablette virtuelle (AVD)

1) Relancer tools/android

2) Menu Tools, item Manage AVDs...

    → fenêtre de gestion des AVDs : AVD manager

3) Bouton new...

    → fenêtre de création d'un AVD

- Name = MonPremierAVD
- Target = Android 2.1 ou 2.2 selon votre SDK

4) Retour dans gestion des AVDs : bouton start...

- C'est long à lancer...

Mode d'emploi d'un AVD



- Souris = doigt
- Touche Home = écran de base
- Touche F2 = menu
- Touche esc = retour

- Première manip : configurer la tablette en français !

F2, Settings, Language

1er bug : clavier en japonais ?

- Si le clavier virtuel écrit en japonais, c'est un bug,

  → il faut cliquer longuement sur n'importe quelle zone de saisie de texte et choisir le mode de saisie « Clavier Android »

- Type de problèmes (bug) très/trop fréquents avec Android, chercher aide sur http://stackoverflow.com



Communiquer avec la cible : ADB

- Un outil appelé platform-tools/adb (Android Device Bridge) permet de communiquer avec l'AVD ou la tablette connectée par USB
  - sorte de telnet : adb shell
  - sorte de tftp : adb push et adb pull

Commandes de ADB

- **adb devices** : liste les tablettes connectées et reconnues : n° et nom

  → option -s n° pour désigner une tablette précise

- **adb shell** : ouvre un shell (sh) sur la tablette
  - Arbre Unix (/, /etc, …) restreint et très protégé
  - Les applis android sont dans /data/data
  - Les fichiers sdcard sont dans /sdcard
  - Mais peuvent changer selon la tablette
- **adb push** *fichier nomcompletsurtablette*
- **adb pull** *nomcompletsurtablette*

ADB avec une vraie tablette

- Le pb initial : faire accepter votre tablette par votre système !
  - Il y a le mode 'support de stockage' : ok sur tous les système, la tablette apparaît comme un disque
  - Et le mode 'adb', c'est celui qui pose pb... (driver)
    - C'est le mode qu'on active sur une vraie tablette par le menu Paramètres – item Applications – sous-item Développement – ss-item Débogage USB

http://developer.android.com/tools/device.html

Une vraie tablette sur Linux

- Sur Linux, quand on branche un périphérique USB, c'est **udev** qui le fait apparaître dans le système d'exploitation
  - sous la forme d'un /dev/fichier spécial

    http://fr.wikipedia.org/wiki/Udev

    http://doc.ubuntu-fr.org/udev
- Encore faut-il que ses identifiants USB vendor:product soient connus et autorisés

  => « règle » à définir

ADB sur Linux (partie 1)

- Commande lsusb

  → Repérer les identifiants hexa de la tablette, ex : 0e79:1411

- sudo gedit /etc/udev/rules.d/51.android.rules

  → mettre SUBSYSTEM=="usb", ATTRS{idVendor}=="*0e79*", ATTRS{idProduct}=="*1411*", MODE="0666"

- sudo chmod a+r /etc/udev/rules.d/51.android.rules
- sudo restart udev

ADB sur Linux (partie 2)

- Pour finir, il faut faire connaître votre tablette au SDK :

  echo "0x*0e79*" >> ~/.android/adb_usb.ini

- Pour tester : adb devices

  → doit afficher la liste des tablettes connectées

  - Si la liste reste vide, c'est qu'il y a un problème (stackoverflow ou fed up!)

ADB sur Windows

- Pour une tablette exotique (Archos), consulter http://developer.android.com/tools/extras/oem-usb.html
- Avant de la brancher la 1e fois, éditer le fichier :

    android-sdk-windows\extras\google\usb_driver\android_winusb.inf

- Trouver ou rajouter une section (ex : archos) :

    [Google.NTamd64]

    ...

    ; Archos

    %CompositeAdbInterface% = USB_Install, USB\VID_*0E79*&PID_*1407*&MI_01

- Puis installer le driver manuellement

Eclipse et Android

Il faut rajouter un plugin nommé Android Development Tools (ADT) dans Eclipse pour accéder à Android

1) Télécharger le plugin
2) Configurer le plugin
3) Le garder à jour par rapport au SDK

Installation du plugin pour Eclipse

a) Dans Eclipse, menu Help, item Install New Software.

b) Bouton Add.. en haut à droite

c) Fenêtre « Add site » : nom="Android Plugin" et Location= https://dl-ssl.google.com/android/eclipse/

d) Cliquer sur OK

e) Work with « Android Plugin », cocher « Developer Tools » et cliquer sur Next

f) Liste des outils => cliquer sur Next, accepter les licences.

Configuration du plugin Eclipse

2) Au redémarrage d'Eclipse, fournir le chemin d'accès au SDK Android
   - Là où on trouve tools et platform-tools
   - Il se trouve aussi dans le dialogue des préférences, onglet Android
3) Si vous mettez à jour votre SDK, il faut aussi mettre à jour votre ADT <u>et réciproquement</u>
   - Eclipse, menu Help, Check for updates...
   - NB : le SDK est mettable à jour très souvent... lancer tools/android

Programmation Android

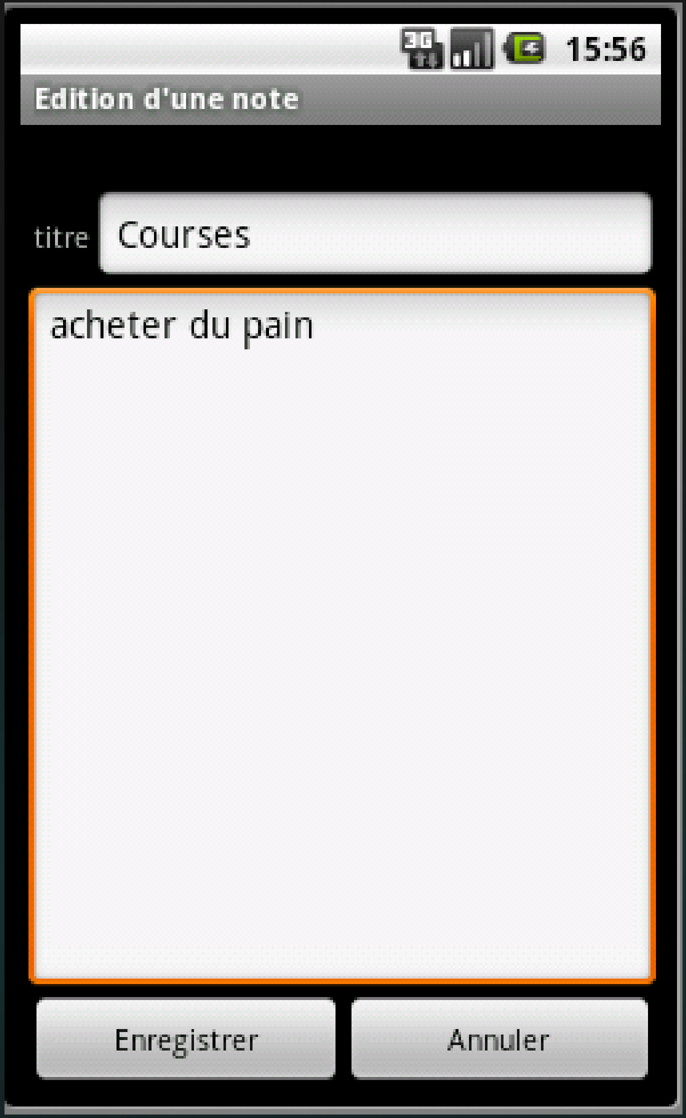
Activité = classe contenant les fonctions pour gérer l'une des pages de l'application

Ressources = textes, images

Manifeste = déclaration du logiciel

Application Android

- Une application Android est composée d'activités
- Une activité = 1 page, avec des contrôles (TextView, EditText, Button...)
- Android gère une sorte de pile d'activités, seule celle du premier plan est active
  - Bouton retour => retour à la précédente
  - Ok => fin de l'activité



Déclaration des activités

- Toutes les activités d'un logiciel doivent être déclarées dans le fichier AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="fr.iutlan.tutoandroid" ...>
  <application ...>
    <activity android:name=".TutoAndroidMainActivity" />
    <activity android:name=".TutoAndroidAboutActivity" />
    ...
  </application>
</manifest>
```

Types d'activités

- Certaines activités sont marquées, par exemple pour être démarrables de l'extérieur :

```
<activity android:name=".TutoAndroidMainActivity" ... >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

- Le nom « intent filter » vient du fait que c'est un « Intent » qui lance une activité, voir plus loin

La classe Activity côté prog

- android.app.Activity
  http://developer.android.com/reference/android/app/Activity.html
- Elle gère le déroulement d'une page :
  - Lancement, arrêt, redémarrage...
- C'est une classe générale qui possède des variantes pour certains types de pages :
  - ListActivity, TabActivity...

La classe Activity (suite)

- On doit la sous-classer :

```
class MonActivite extends Activity {
```

- et surcharger certaines de ses méthodes :
  - Ex : la création de la page se fait par :

```
@Override
public void onCreate(...) {
```

Exemple d'activité

```
package fr.iutlan.tutoandroid;
import android.app.Activity;
...
public class TutoAndroidActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Méthodes d'Activity

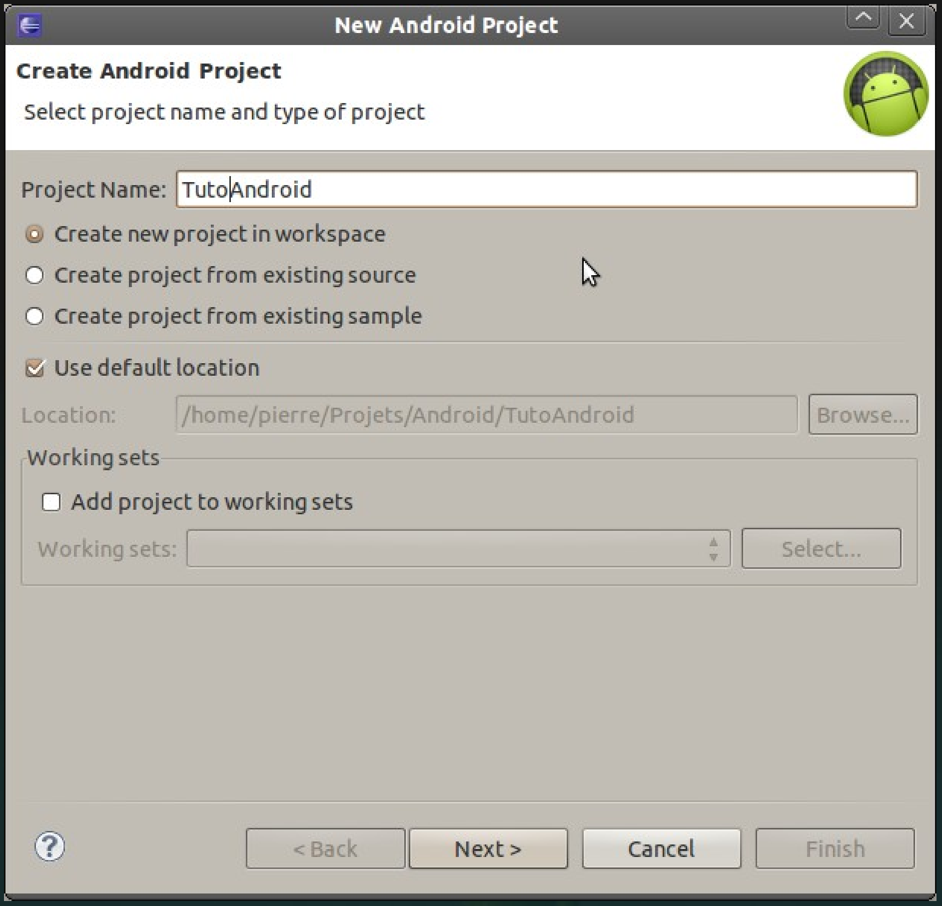
- onCreate : appelée quand on lance l'activité pour la première fois : il faut afficher son interface et initialiser ses variables
  - Ne pas s'occuper, pour l'instant, des paramètres
- onDestroy : appelée quand l'activité est supprimée, d'elle-même ou par manque de mémoire ou à la demande de l'utilisateur (gestion des applications)

Concrètement...

Création du premier projet Android

Créer un projet dans Eclipse

- Dans le menu Fichier, New, Android Project
- Project Name = TutoAndroid
- Vérifier où il est créé
- Bouton Next>

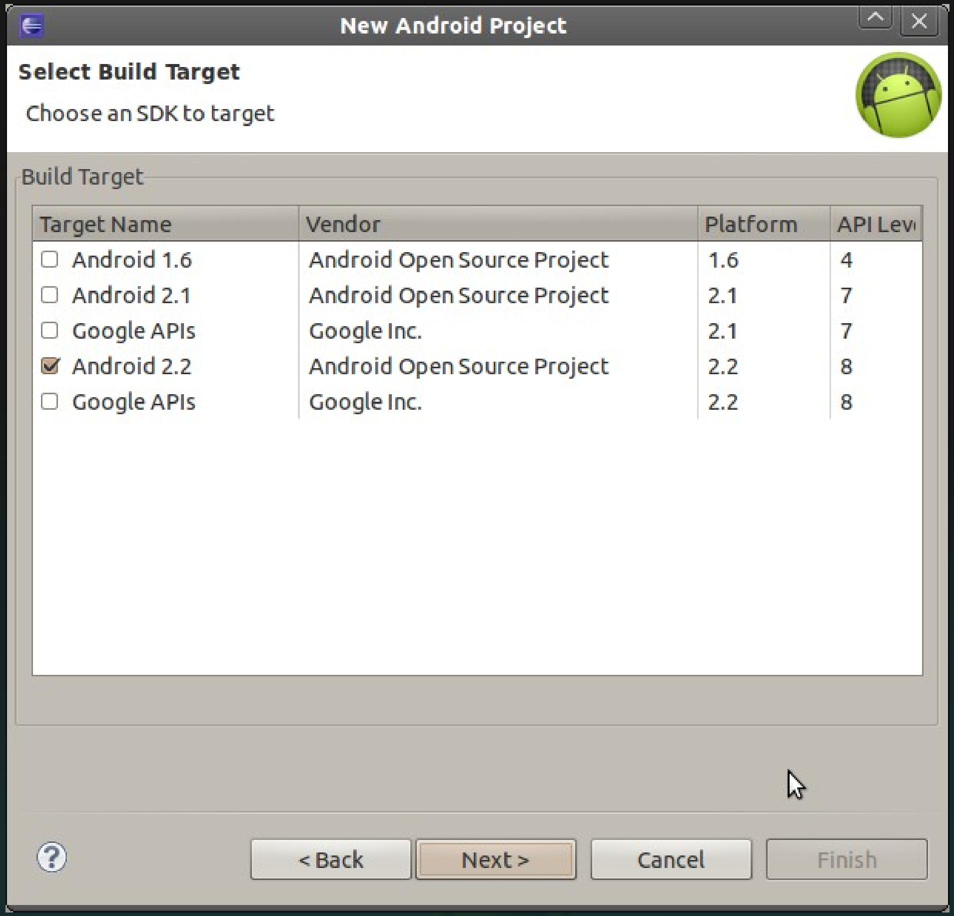


- NB : ce déroulement est celui du plugin pour Eclipse Galileo... Adaptez-le à votre version.

Créer un projet (suite)

- Choisir le SDK
- Bouton Next>

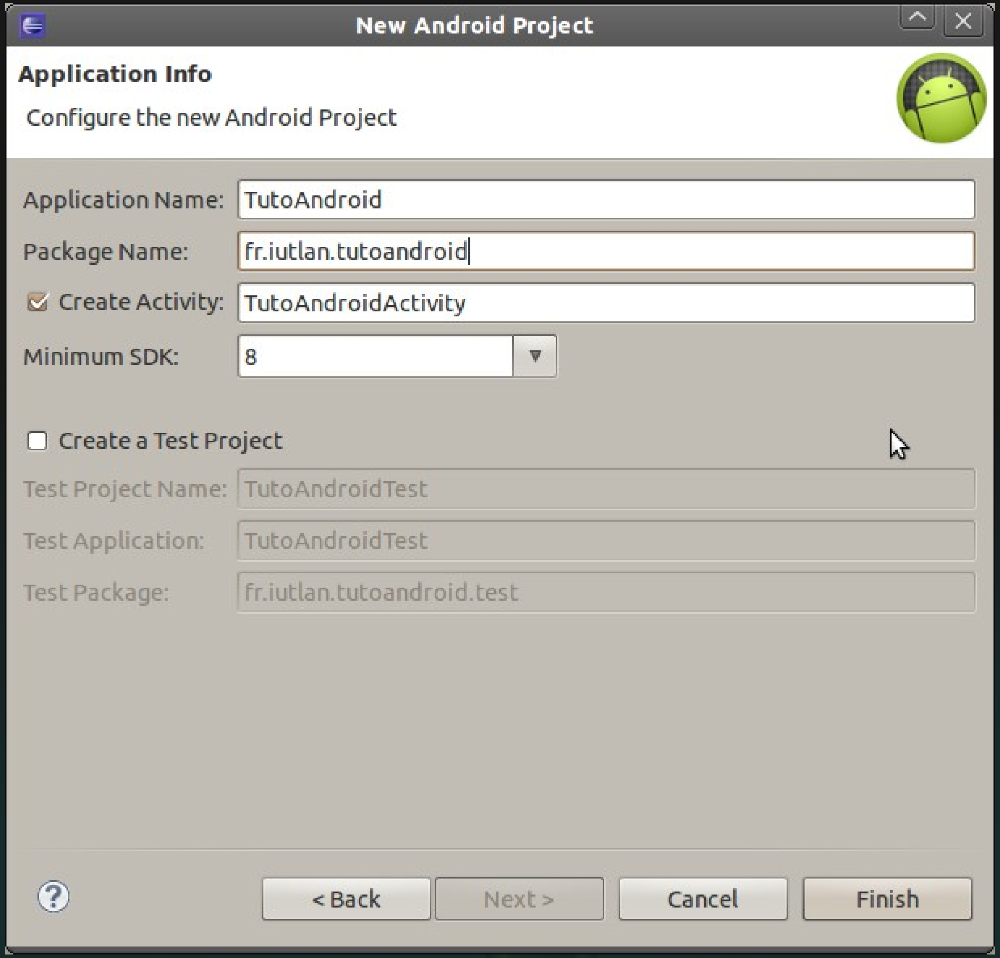
NB : dans le plugin pour Eclipse Indigo, cette page apparaît sous forme d'une liste déroulante directement dans la première page.



Créer un projet (fin)

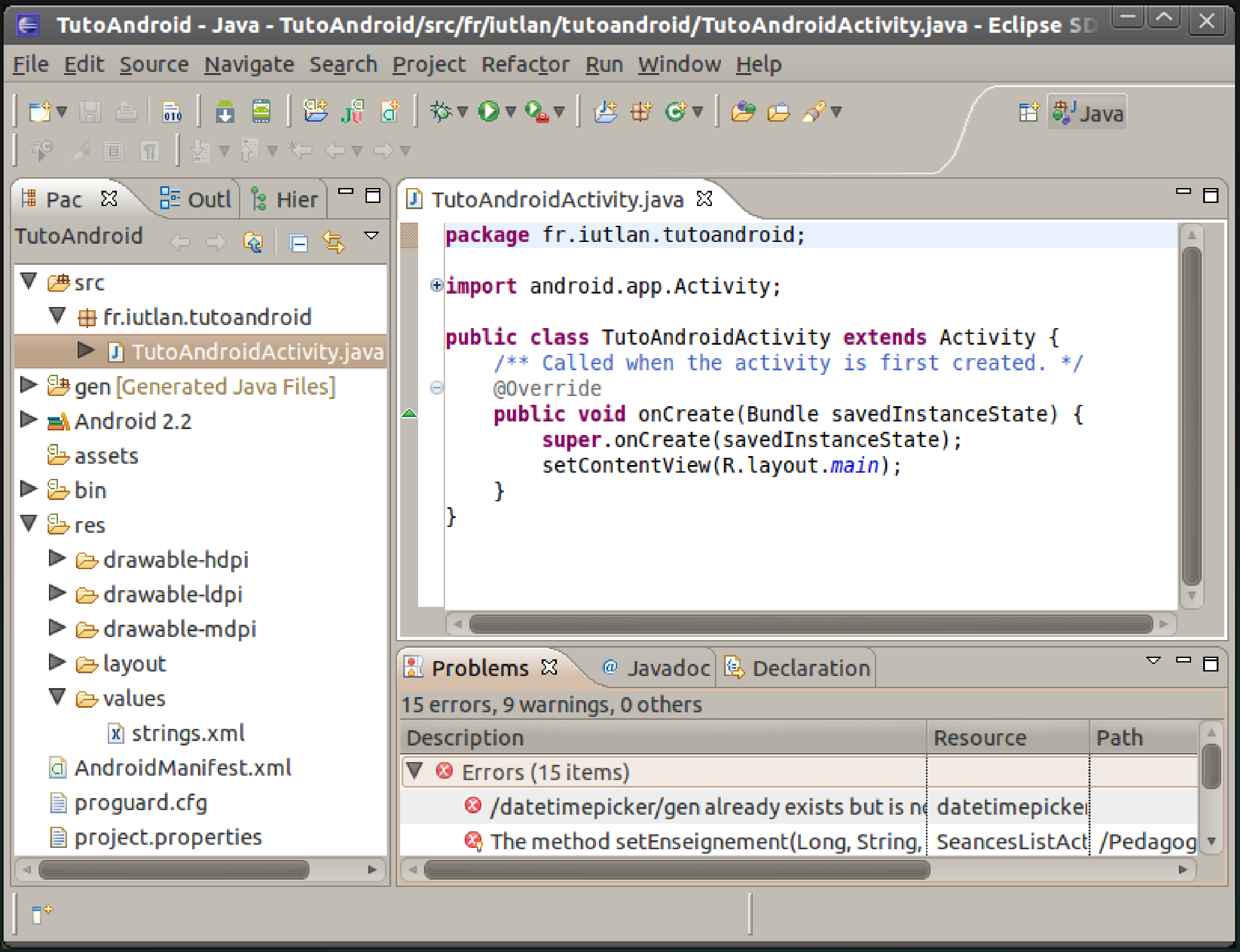
- Choisir le nom de l'application et de l'activité principale
- Choisir le nom du package
  - C'est le nom inversé de l'organisation qui gère le projet, ex :

    fr.iutlan.tutoandroid
- Bouton Finish



Projet créé

- Résultat :

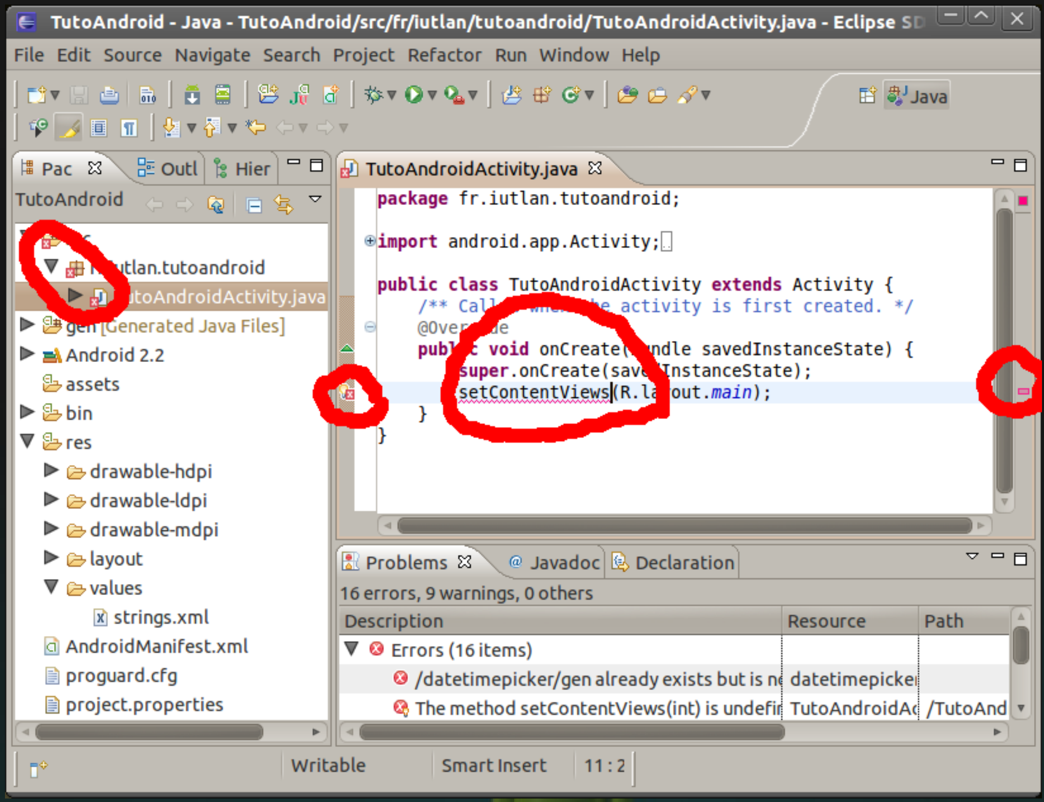


Structure des fichiers d'un projet

- Un projet = arbre de nombreux fichiers
  - src = les sources, organisés en packages
  - res = les ressources, on verra plus tard
  - gen = des sources générés automatiquement à partir des ressources
  - bin = les .class issus des compilations de src et gen
  - assets = d'autres ressources, on ne verra pas
  - Android2.x = les librairies du SDK utilisé
  - AndroidManifest.xml
  - *.properties = config pour Eclipse

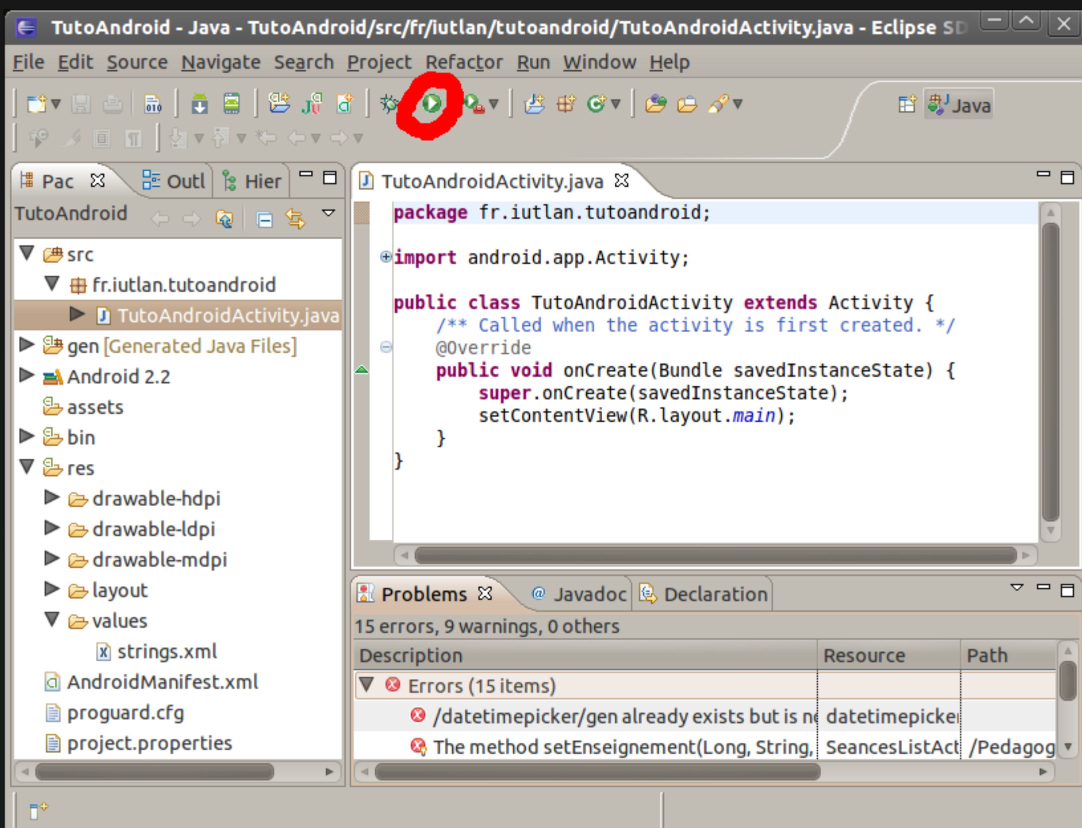
Compiler le projet

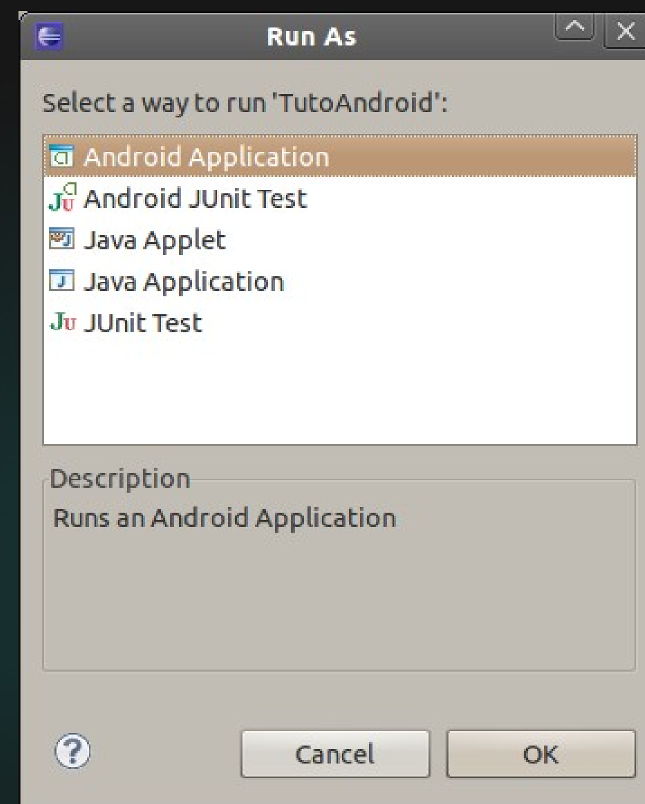
- Pas besoin de compiler, c'est fait en permanence par Eclipse
- Tant qu'il y a des erreurs, il y a du rouge sur les fichiers et on ne peut pas lancer



Lancer le projet

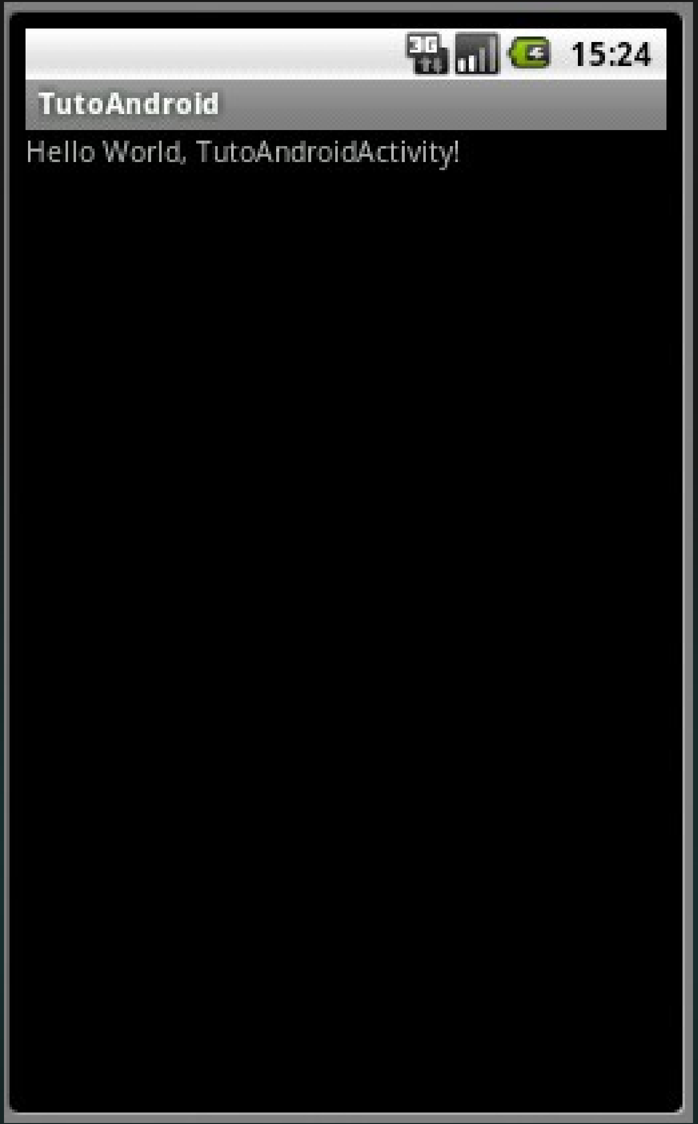
- Cliquer sur la flèche verte ou choisir le menu « Run As… » et, la première fois, choisir Android Application





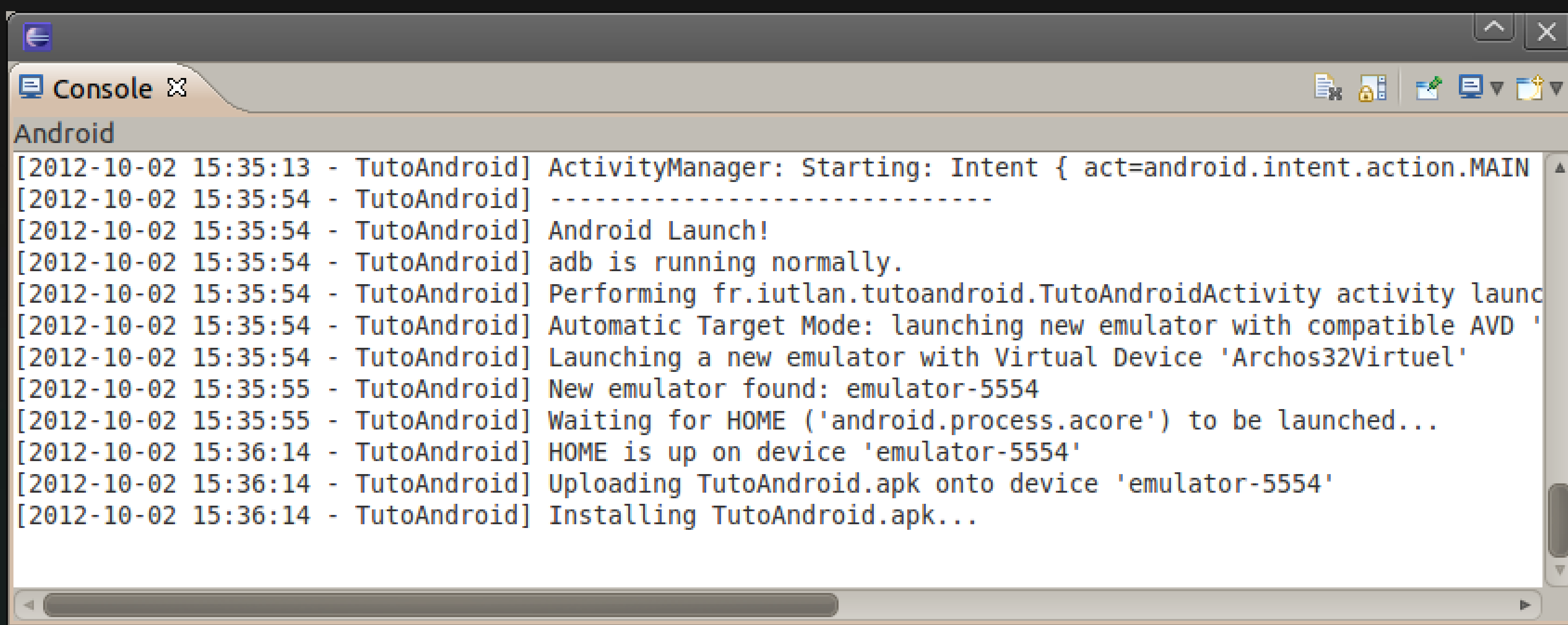
Lancement d'un projet

- Eclipse lance un émulateur ou utilise la tablette branchée
- Installe puis lance le logiciel



Suivi du lancement

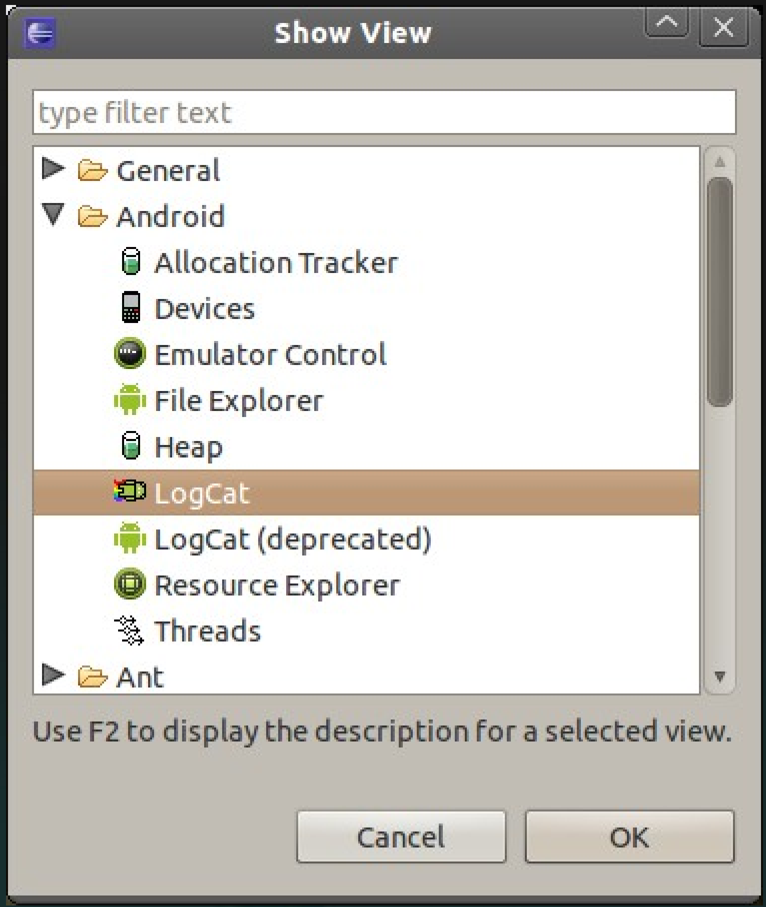
- Les étapes du lancement sont affichées dans la fenêtre console (dans les versions Indigo, il y a du rouge sur certaines lignes)



Suivre l'exécution : LogCat

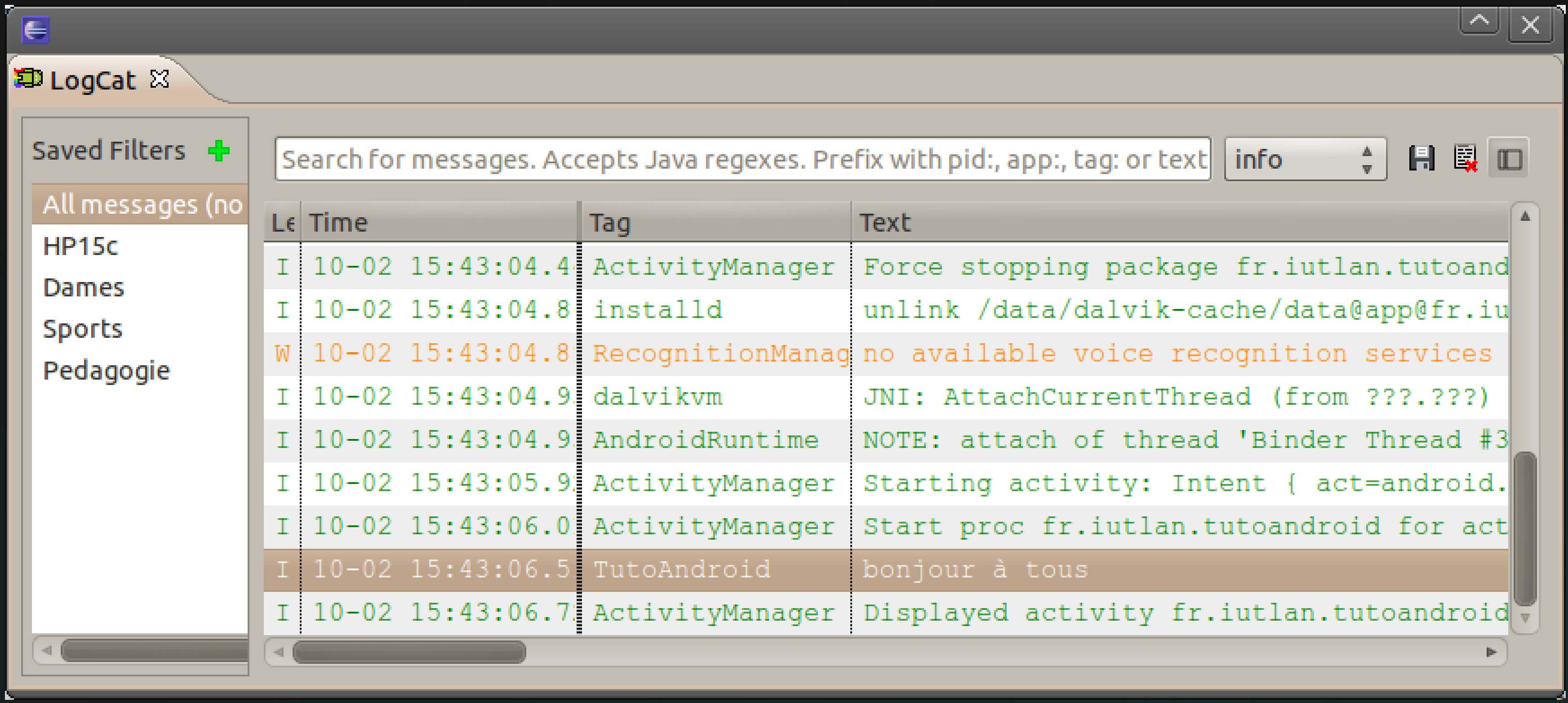
- D'abord, afficher ou activer la fenêtre Devices
  - menu Window, item Show View, sous-item Other...
  - Dans la liste, choisir la tablette active
- Afficher la fenêtre LogCat

  => LogCat se remplit avec les messages de la tablette sélectionnée



LogCat : la tablette vous parle

- LogCat permet de communiquer avec la tablette : tout message qu'elle émet est capté



LogCat : mode d'emploi

- Deux colonnes utiles :
  - TAG = nom court du logiciel émettant un message
  - Text = texte émis
- Filtrages par niveau de gravité et TAG
- Émission de messages : le package android.util.Log contient la classe Log :
  - Log.i(TAG, Text); affiche un message d'information
  - Log.w(TAG, Text); message d'avertissement
  - Log.e(TAG, Text); message d'erreur

Exemple : afficher un message

- Dans TutoAndroidActivity.java, après la ligne

```
super.onCreate(savedInstanceState);
```

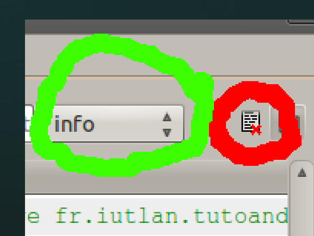
  rajouter la ligne suivante :

```
Log.i("TutoAndroid", "Salut à tous !");
```

- Eclipse voudra/devra rajouter au début :

```
import android.util.Log;
```

- Relancez et regardez LogCat, il y a votre message dans la longue liste
  - on peut rajouter des filtres
  - Penser à effacer le log avant un lancement



Trouver un bug avec LogCat

- LogCat sert à localiser la source des bugs.
- Rajouter les 2 lignes suivantes dans la méthode onCreate de votre activité

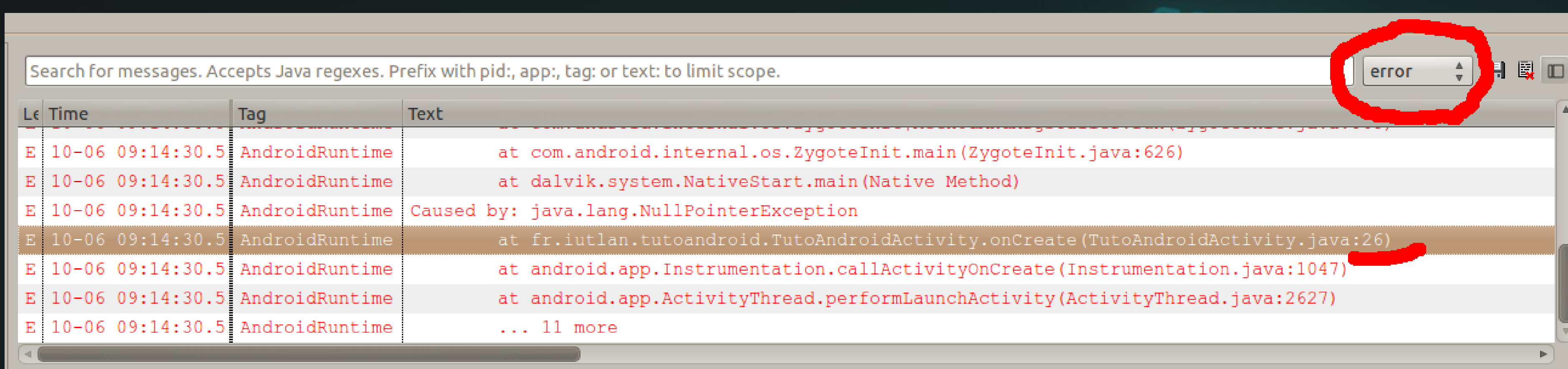
```
TextView tv2=null;
tv2.setText("boum");
```

  - Ignorer le warning



Trouver la faille

- Ouvrir l'onglet LogCat, filtrer sur les erreurs
- Double-cliquer sur la ligne qui concerne votre thread et votre package :
  - 1 appli Android = plusieurs threads
  - 1 thread = plusieurs fonctions imbriquées, l'erreur peut apparaître dans la cave mais causée en cuisine



Les ressources

Ressources = interfaces, textes, images du logiciel
certaines sous forme déclarative dans un fichier XML

D'où vient le message affiché ?

- Ouvrir le fichier res/**values/strings**.xml

  NB : choisir la vue source XML du fichier (onglet en bas de la fenêtre)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World !</string>
    <string name="app_name">TutoAndroid</string>
</resources>
```

- Les chaînes sont placées entre des balises <string> et identifiées par l'attribut name.

Utilisation des ressources string

- Dans un autre fichier XML :
  - Ex : dans le manifeste

    <activity **android:label="@string/app_name"**
  - La notation @string/xxx référence la chaîne name="xxx" dans res/values/strings.xml
- Dans un source .java :
  - Voir gen/package/R.java : chaque chaîne est associée à une constante entière, son identifiant
  - La classe R est générée automatiquement
  - BUG! parfois elle est mal regénérée ! → nettoyer le projet : menu project, item Clean...

Autre type de ressources

- Ouvrir le fichier res/**layout**/main.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ... >
   <TextView ...
      android:text="@string/hello" />
</LinearLayout>
```

- Un layout décrit la mise en page de l'activité
- On retrouve la ressource <string name="hello">
- Exercice : changer le texte de cette ressource

Référencement d'un layout

- Ouvrir src/package/TutoAndroid.java :

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.i("TutoAndroid","bonjour à tous");
    setContentView(R.layout.main);
}
```

- La classe gen/package/R.java contient la constante R.**layout.main** qui identifie res/**layout/main**.xml
  - celui-ci spécifie le contenu de la page (content view)

Autre type de ressource

- Ouvrir le dossier res/drawable-hdpi (ou mdpi...)
  - Il y a une image nommée ic_launcher.png
- Son nom de base l'identifie et on la retrouve dans AndroidManifest.xml :

  <application

  android:icon="**@drawable/ic_launcher**"

  android:label="@string/app_name" >

- Remarquer que dans R.java, il y a un final int R.drawable.ic_launcher

Manip sur les ressources

1) Dans strings.xml, rajouter une ressource

```
<string name="blabla">voici du blabla</string>
```

remarquer que R.java a été mis à jour

2) Dans main.xml, rajouter un attribut au TextView :

```
<TextView android:id="@+id/textview"
```

la syntaxe @+id/xxx fait *créer* un identifiant, tandis que @id/xxx le référence seulement

remarquer que R.java a été mis à jour (sinon clean...)

- On va faire afficher blabla dans le textview

Manip sur les ressources (suite)

3) Dans TutoAndroid.java, rajouter ceci après setContentView(R.layout.main) :

```
TextView tv =
    (TextView)this.findViewById(R.id.textview);
tv.setText(R.string.blabla);
```

4) Relancer l'application

- On constate qu'on a remplacé le texte initial par celui de la ressource blabla

NB : parfois il faut nettoyer le projet pour reconstruire le fichier R.java (bug ADT/Eclipse), en particulier si on rajoute des ressources (problème assez fréquent)

Bilan de cette manip

- Cette ligne :

  (TextView)this.**findViewById**(R.id.textview)

  - s'applique à l'Activity : elle recherche une View (contrôle, widget, composant... de l'interface) ayant cet identifiant
  - Elle retourne null ou le TextView en question
  - Il faut convertir le type du résultat
- Cette ligne : tv.**setText**(R.string.blabla);
  - est une méthode de TextView et lui change son libellé avec un numéro de ressource string

Vues et méthodes

- Il existe un grand nombre de types de vues
  - Textview, EditView, Button, Spinner, CheckBox...
  - Voir le package android.widget

http://developer.android.com/reference/android/widget/package-summary.html

- Chacune a un grand nombre de méthodes, spécifiques ou héritées
  - Exemple, la classe Button :

http://developer.android.com/reference/android/widget/Button.html

- En général, tous les setters et getters correspondant aux variables membres.

Setters et attributs xml

- On retrouve une grande partie des setters Java parmi les attributs utilisables dans les ressources XML
  - Exemples pour un TextView :
    - setText("...") correspond à android:text="..."
    - setLines(nb) correspond à android:lines="nb"
- Exemple :

```
<TextView android:id="@+id/textview"
   android:textSize="15sp"
   android:textColor="#F0F"
```

- Pour les connaître, une seule source : la doc

Ressources ou valeurs en dur ?

- On peut généralement placer les constantes dans les ressources et fournir leurs identifiants ou directement les constantes en dur :
  - tv.setText(R.string.hello); // voir res/values/strings.xml
  - tv.setText("Salut !"); // hard-coded
- L'avantage des ressources est :
  - extérieures au logiciel, traductibles et éditables sans recompilation
- Inconvénient :
  - plus lourd à mettre en œuvre

Layouts

Layout = description de la mise en page

Ajout d'un bouton dans la fenêtre

- Dans res/layout/main.xml, rajouter les lignes suivantes avant le </LinearLayout> final :

```
<Button android:id="@+id/bouton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="OK" />
```

- Vous pouvez aller vérifier le résultat dans l'onglet GraphicalLayout (bas de cette fenêtre)
- NB : le libellé du bouton a été écrit en dur

Ajout d'un bouton (suite)

- Si vous lancez le logiciel, vous verrez que le bouton s'affiche mais n'est pas actif
- Voici comment le rendre vivant :
  - Il faut lui associer un listener, c'est à dire un objet qui saura réagir si on appuie sur le bouton
  - Le listener possède une méthode onClick qui est appelée quand on appuie sur le bouton

Ajout d'un listener au bouton

- Association d'un listener au bouton :

```
Button btn = (Button)findViewById(R.id.bouton);
btn.setOnClickListener(this);
```

- Le choix a été fait, ici, que le listener soit l'activité elle-même : this !
  - Voir plus loin pour une autre approche
- Il faut donc qu'elle définisse la méthode **onClick**

Définition du Listener

- Éditer l'entête de la classe :

```
public class TutoAndroidActivity extends Activity
implements OnClickListener {
```

- Rajouter la méthode :

```
@Override
public void onClick(View btn) {
    TextView tv = (TextView)findViewById(R.id.textview);
    tv.setText("Pouic");
}
```

Gestion des vues de l'interface

- Normalement, on garde une copie des View utilisées, en tant que variables membres :

```
public class TutoAndroidActivity ... {
    // variables membres globales
    TextView tv;
    Button btn;
    public void onCreate(...) {
        tv = (TextView)findViewById(R.id.textview);
        btn = (Button)findViewById(R.id.bouton);
        ...
```

Deux boutons pour le prix d'un

- Exercice : maintenant, rajoutez un autre bouton
  - Identifiants : bouton1 et bouton2
- Le premier doit afficher « Oui », l'autre « Non » dans le TextView
- Problème : on doit associer le même listener, this, aux deux boutons, donc impossible de les distinguer, sauf si...

Solution 1 : switch sur l'identifiant

```
@Override
public void onClick(View btn) {
    switch (btn.getId()) {
    case R.id.bouton1:
        tv.setText("Oui");
        break;
    case R.id.bouton2:
        tv.setText("Non");
        break;
    }
}
```

Solution 2 : plusieurs listeners

- Au lieu de : btn2.setOnClickListener(this);
- mettre ceci (pareil pour btn1) :

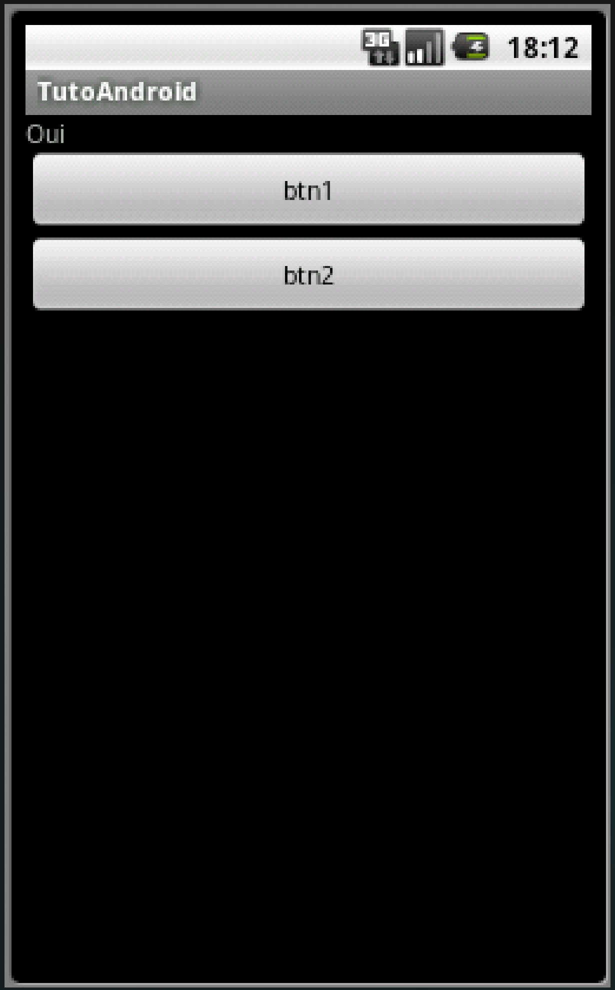
```
Button btn2 = (Button)findViewById(R.id.bouton2);
btn2.setOnClickListener(
    new OnClickListener() {
        @Override
        public void onClick(View v) {
            tv.setText("Non");
        }
});
```

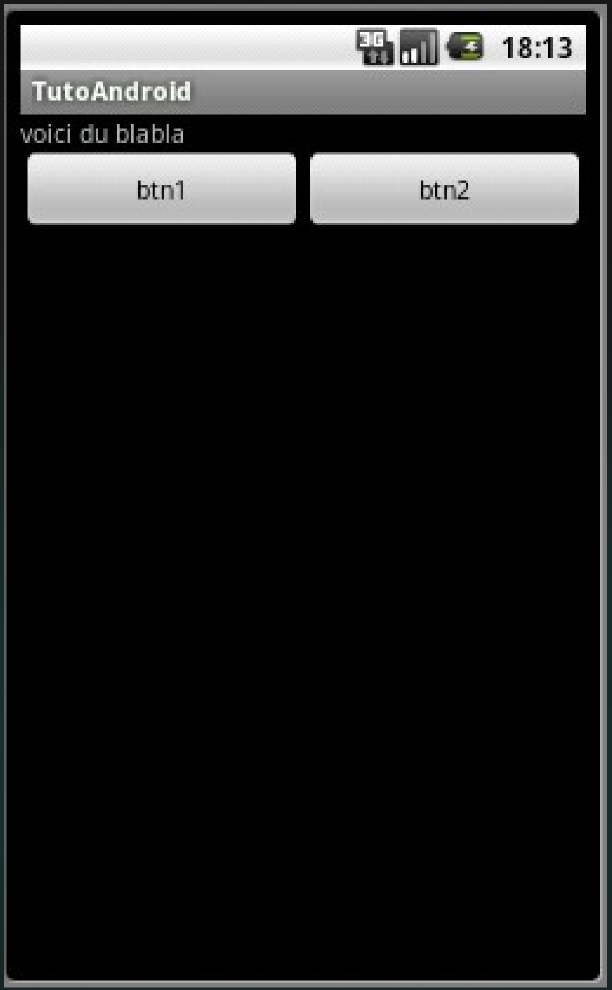
Comparaison des deux solutions

- La solution 2 semble plus simple
  - 1 listener par bouton (et par type d'action)
  - Accès aux variables globales
  - Mais pb pour accéder aux variables locales de la méthode qui installe le listener
    - => **final** int varlocale = parametre;
- De toutes façons, pour les menus, on retrouvera la solution 1, la seule possible

On revient sur la disposition

- Au lieu de ceci : on voudrait cela :





Layouts

- Les vues (contrôles graphiques) doivent être arrangées dans des containers appelés Layouts
  - LinearLayout : arrange les vues séquentiellement selon un axe horizontal ou vertical
  - RelativeLayout : arrange les vues comme on veut l'une par rapport à l'autre (pas vu aujourd'hui)
- Un layout se comporte comme un parent pour les vues qu'il contient : ses enfants

Taille d'une vue

- Une vue reçoit de son parent une largeur et une hauteur à partir desquels elle définit sa propre taille :
  - android:layout_width
  - android:layout_height
- Elle peut choisir de l'occuper ou pas :
  - android:layout_width="**fill_parent**"

    NB : fill_parent est devenu **match_parent** dans le SDK 2.2
  - android:layout_width="**wrap_content**"

Compétition entre enfants

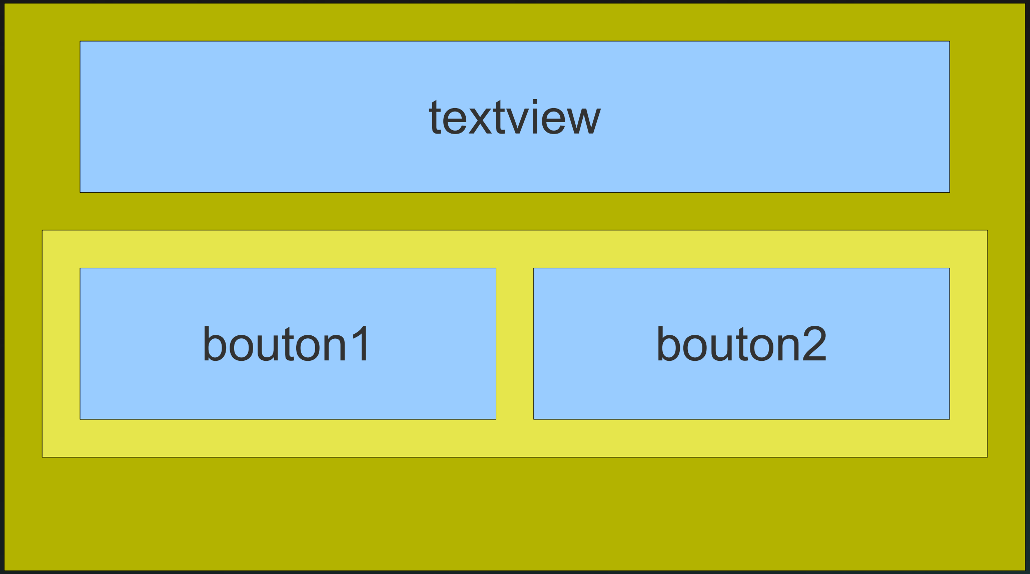
- En cas de compétition entre enfants, par exemple quand les deux prétendent occuper toute la place :

```
<Button android:id="@+id/bouton1"
  android:layout_weight="1"
  android:layout_width="fill_parent" ... />
<Button android:id="@+id/bouton2"
  android:layout_weight="1"
  android:layout_width="fill_parent" ... />
```

- On leur rajoute un poids pour équilibrer leurs demandes

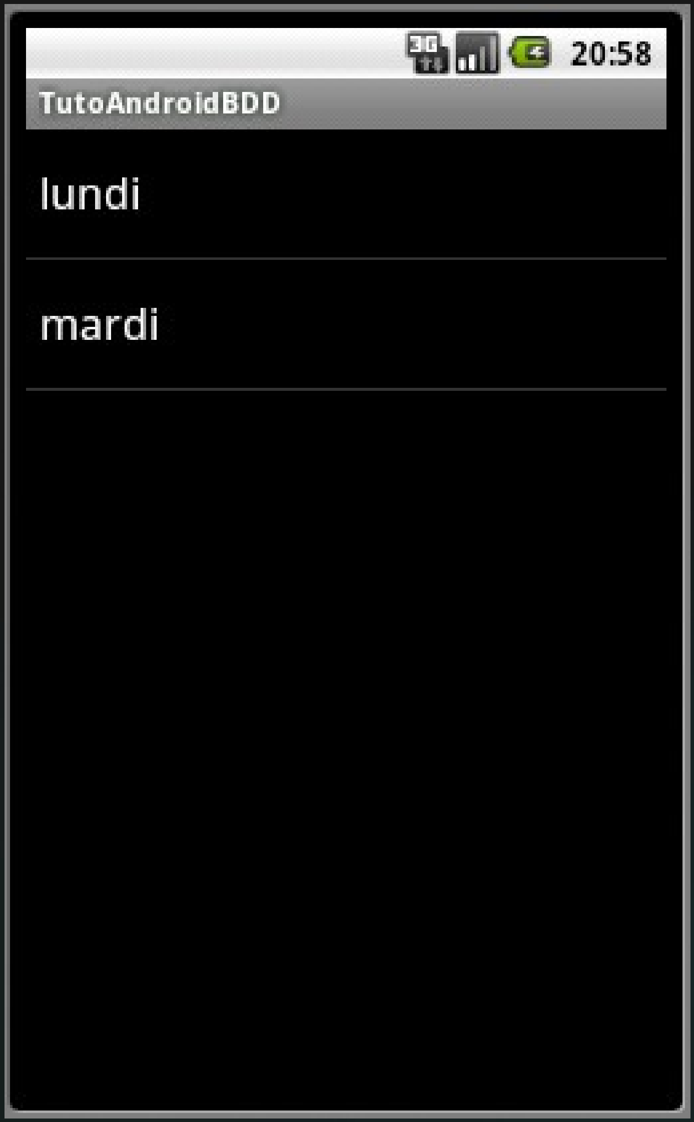
Pour arriver à nos fins

- Voici l'imbrication qu'il faut obtenir :



- En jaune, deux LinearLayouts, l'un horizontal, l'autre vertical
- Les deux boutons doivent se partager la place

Android et SQL



On passe à une application qui affiche une liste d'items tirés d'une base SQL

Principes

- On va utiliser une sous-classe d'Activity : ListActivity
- Elle intègre un layout contenant un ListView
- La fonction d'un ListView est de visualiser une liste d'items fournis par un « fournisseur de données »
- Pour cela, on utilise un « adaptateur de BDD » associé à une requête SQL Select ... from...
- Une telle requête est gérée par un « Curseur »

Base de données SQL sur Android

- Android intègre un SGBD SQLite v3
- La base de données d'une application est stockée dans /data/data/package/nom.db
- Pour bien gérer une bdd dans une application android, il faut créer deux classes :
  - Classe BDD qui représente la base ouverte
    - ses méthodes = requêtes SQL + vérifications
  - Classe BDDHelper qui aide à ouvrir une base :
    - sous-classe de SQLiteOpenHelper
    - sa méthode onCreate = crée les tables au tout début

C'est parti !

- Créer un nouveau projet Android
  - Nom = TutoAndroidBDD
  - On changera la classe Activity plus tard
- Créer une classe BDDHelper dérivée de SQLiteOpenHelper :
  - Menu File, New, Class...
  - Nom = BDDHelper
  - Superclasse = android.database.sqlite.SQLiteOpenHelper

    http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html
  - Cocher créer les méthodes virtuelles héritées

Constructeur de BDDHelper

- Rajouter un constructeur :

```
public BDDHelper(Context context) {
    super(context, "infos.db", null, 1);
}
```

- Context est une superclasse de Activity
- On fournit le nom de la base et sa version

Méthode onCreate de BDDHelper

- Elle doit créer les tables de la base :
  - @Override
    public void onCreate(SQLiteDatabase db) {
      db.execSQL("CREATE TABLE infos (_id INTEGER PRIMARY KEY AUTOINCREMENT, info TEXT NOT NULL)");
    }
  - execSQL(String requete) pour toute requête sauf SELECT
- Ensuite, on peut remplir avec les données initiales

Ajout de données initiales

- Ajouter les lignes suivantes dans la méthode onCreate de BDDHelper

```
db.execSQL("INSERT INTO infos VALUES (1, 'lundi')");
db.execSQL("INSERT INTO infos VALUES (2, 'mardi')");
```

- Attention, ces instructions ne sont faites qu'une seule fois dans la vie du logiciel sur une tablette : lors de son premier lancement
- D'autre part, on ne doit pas faire comme ça (mais pour simplifier..., voir méthode insert plus loin)

Classe BDD

- La classe BDDHelper n'est qu'une aide pour l'ouverture et éventuellement la création de la base de données
- Il faut une classe pour simplifier la gestion de la base : la classe BDD
- Dans l'activité principale, on fera :

```
BDD mabase = new BDD();
mabase.open();
```

  - Et automatiquement la première fois seulement, ça va appeler le helper pour créer la base.

Classe BDD (suite)

- Créer une classe appelée BDD :

```
public class BDD {
    BDDHelper helper;        // le helper pour l'ouverture/création
    SQLiteDatabase base;    // l'objet qui représente la base

    public void open(Context activity) throws SQLException {
        // créer ou ouvrir la base de données
        helper = new BDDHelper(activity);
        base = helper.getWritableDatabase();
    }
}
```

Base et Activité

- Voici comment l'activité ouvre la base :

```
public class TutoAndroidBDDActivity extends ... {
    BDD bdd;
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bdd = new BDD();
        bdd.open(this);
    }
```

- bdd est une variable globale qui reste affectée une fois l'activité créée

Requêtes SQL et curseurs

- Pour interroger la base, on crée un curseur :
  - Définir **Cursor cListeInfos;** en membre global
  - À la suite de bdd.open(this); rajouter ceci :

```
cListeInfos = bdd.getInfos();
startManagingCursor(cListeInfos);
```

- Il reste à définir la méthode getInfos dans BDD :

```
public Cursor getInfos() {
    return base.rawQuery(
        "SELECT _id, info FROM infos", null);
}
```

Deux remarques importantes

- Quand on crée un curseur par un SELECT, il est essentiel d'extraire un identifiant appelé _id pour les données

  SELECT _id, ... FROM …

  - Il doit être placé en premier
  - Même si cet identifiant ne sert pas ensuite
- Dans android2.1, les attributs sont nommés table.nom, dans 2.2, seulement nom (!)

Transformer l'activité

- On va maintenant migrer vers une activité de type ListActivity
- Cela impose des changements dans :
  - La classe principale
    - Changer sa superclasse
    - Rajouter des méthodes de gestion de la liste
  - Le fichier de layout
    - Utiliser un layout prédéfini ou le notre

Changer la superclasse et le layout

- Commencer par mettre ListActivity au lieu de Activity dans TutoAndroidBDD

```
import android.app.ListActivity;
public class TutoAndroidBDDActivity extends ListActivity {
```

- Changer le layout de l'activité :

```
setContentView(android.R.layout.activity_list_item);
```

ou

```
setContentView(R.layout.main);
```

modifié

Layout android.R.layout.activity_list_item

- C'est l'un des layouts prédéfinis, mais il est défini incorrectement, donc main.xml devient :

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent" android:layout_height="fill_parent">
   <ListView android:id="@android:id/list"
      android:layout_width="fill_parent" android:layout_height="fill_parent"/>
   <TextView android:id="@android:id/empty"
      android:layout_width="fill_parent" android:layout_height="fill_parent"
      android:gravity="center" android:text="- vide -" />
</FrameLayout>
```

- ListActivity repère la liste @android:id/list pour la remplir automatiquement

Composant ListView

- On en arrive au remplissage de la liste à partir de la base de données
- Un ListView qui est la base d'une ListActivity affiche des éléments sous forme de liste
  - Voir http://developer.android.com/guide/topics/ui/layout/listview.html
- Le contenu du ListView est défini par un « Adapter », par exemple un SimpleCursorAdapter pour un Cursor de bdd

SimpleCursorAdapter

- C'est un objet qui associe une requête SQL fournie sous forme d'un curseur à un ListView
- On indique :
  - quel curseur il faut lire
  - quel layout il faut utiliser pour chaque ligne
    - Par exemple android.R.layout.simple_list_item_1
    - Ce layout possède des TextView dont les identifiants sont connus (text1 et text2)
  - quels champs de la requête on doit afficher dans la liste
  - dans quel TextView on met tel champ

Ajout d'un adapter à la liste

- Toujours à la suite dans TutoAndroidBDDActivity.onCreate :

```
SimpleCursorAdapter adapter = new
    SimpleCursorAdapter(this,
        android.R.layout.simple_list_item_1,
        cListeInfos,
        new String[] {"info"},
        new int[] {    android.R.id.text1 });
setListAdapter(adapter);
```

Mise à jour de la liste

- Ces méthodes sont maintenant obsolètes, mais dans les SDK 2.x, elles permettent de gérer l'état de la requête sous-jacente.
  - Raison : une requête peut durer trop longtemps et bloquer le système.
- On avait écrit l'instruction suivante :

```
startManagingCursor(cListeInfos);
```

- Pour mettre à jour la liste affichée :

```
cListeInfos.requery();
```

  - Cela relance le SELECT et met à jour l'adapter, donc la liste affichée.

Ajout d'un listener pour les clics

- Si on veut rendre la liste cliquable :

```
@Override
protected void onListItemClick(ListView l, View v,
int position, long idInfo)
{
    super.onListItemClick(l, v, position, idInfo);
    Log.i("Tuto", "clic sur "+idInfo);
}
```

- La méthode reçoit l'identifiant de l'élément cliqué

Ajout d'un menu

- On veut maintenant rajouter un menu global permettant de créer des éléments dans la liste
- Ça se fait en deux temps :
  - Affichage du menu à la demande de l'utilisateur
  - Réponse à la sélection d'un item

Affichage du menu

- Voici comment on fait dans Tuto...Activity :

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_optionsmenu, menu);
    return true;
}
```

- Ensuite, il faut définir le menu dans un XML
- L'inflater va lire le XML et créer les vues

Définition du menu

- Cela se fait dans les ressources :
  - Créer le dossier res/menu
  - Créer le fichier main_optionsmenu.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_creerinfo"
          android:title="créer" />
</menu>
```

NB : on a mis le libellé en dur

Réaction au menu

- Il reste à capturer les sélections d'items :

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
  switch (item.getItemId()) {
  case R.id.menu_creerinfo:
    // créer une nouvelle info
    … ex : Log.i("Tuto", "on va créer une info");
    break;
  }
}
```

Menu création d'une information

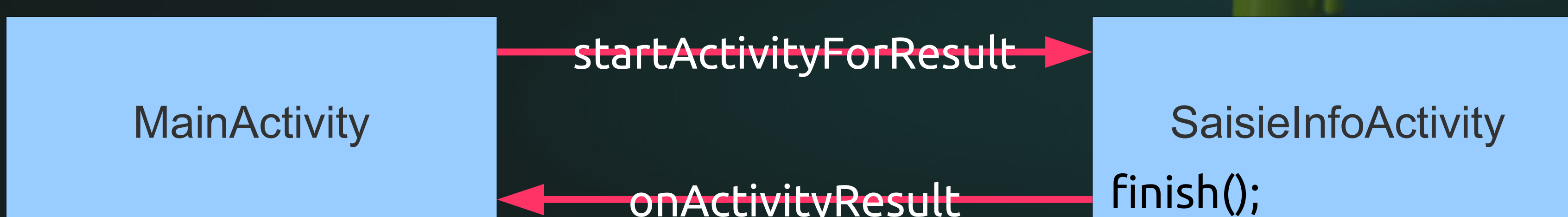
- On souhaite rajouter un n-uplet dans la base
- Il faut demander ses champs à l'utilisateur :
  - Soit un dialogue pop-up : pas aujourd'hui
  - Soit une nouvelle activité (SaisieInfoActivity.java) :
    - cette activité va se lancer devant l'activité actuelle,
    - demander de saisir l'information,
    - l'ajouter à la base de données
    - puis se terminer en prévenant l'activité principale
    - qui se réveillera et mettra à jour la liste affichée.

Lancement d'une activité

- Cela se fait au moyen d'un **Intent**

```
// créer une nouvelle info
Intent intent = new Intent(this,
        SaisieInfoActivity.class);
startActivityForResult(intent, 1);
```

- L'activité actuelle va attendre la fin de la nouvelle :



Classe SaisieInfoActivity

- Créer une nouvelle classe :
  - Nom = SaisieInfoActivity
  - Superclasse = android.app.Activity
- La rajouter dans le Manifeste :

  <activity android:name=".SaisieInfoActivity" />
- On va :
  - Rajouter un layout contenant une zone de saisie
  - Un bouton Ok et son listener
  - Ce listener valide la saisie, crée l'information et quitte

Layout pour l'activité de saisie

- Dans res/layout, créer saisie.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:orientation="vertical" >
    <EditText android:id="@+id/edittext" android:editable="true"
        android:layout_width="fill_parent" android:layout_height="wrap_content"/>
    <Button android:id="@+id/btnok" android:text="OK"
        android:layout_width="fill_parent" android:layout_height="wrap_content"/>
</LinearLayout>
```

SaisieInfoActivity (le layout)

- Voici le début

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.saisie);
```

SaisieInfoActivity (le bouton)

- L'entête de la classe :

```
import android.view.View.OnClickListener;
public class SaisieInfoActivity extends Activity
implements OnClickListener {
```

- Ensuite, on associe le listener du bouton OK :
  - findViewById(R.id.**btnok**).setOnClickListener(this);
- Le listener :

```
@Override public void onClick(View v) {
    // lire le texte et l'ajouter dans la base...
}
```

SaisieInfoActivity (la création)

- Lire la valeur saisie :

```
EditText et = (EditText)findViewById(R.id.edittext);
String info = et.getText().toString();
```

- L'ajouter dans la base :

```
BDD bdd = new BDD();
bdd.open(this);
if (bdd.createInfo(info) > 0)
```

- Quitter l'activité :

```
finish();
```

Insertion d'un n-uplet

- Dans la classe BDD, voici comment le faire proprement :

```
public long createInfo(String info) {
    ContentValues values = new ContentValues();
    values.putNull("_id");
    values.put("info", info);
    return base.insert("infos", null, values);
}
```

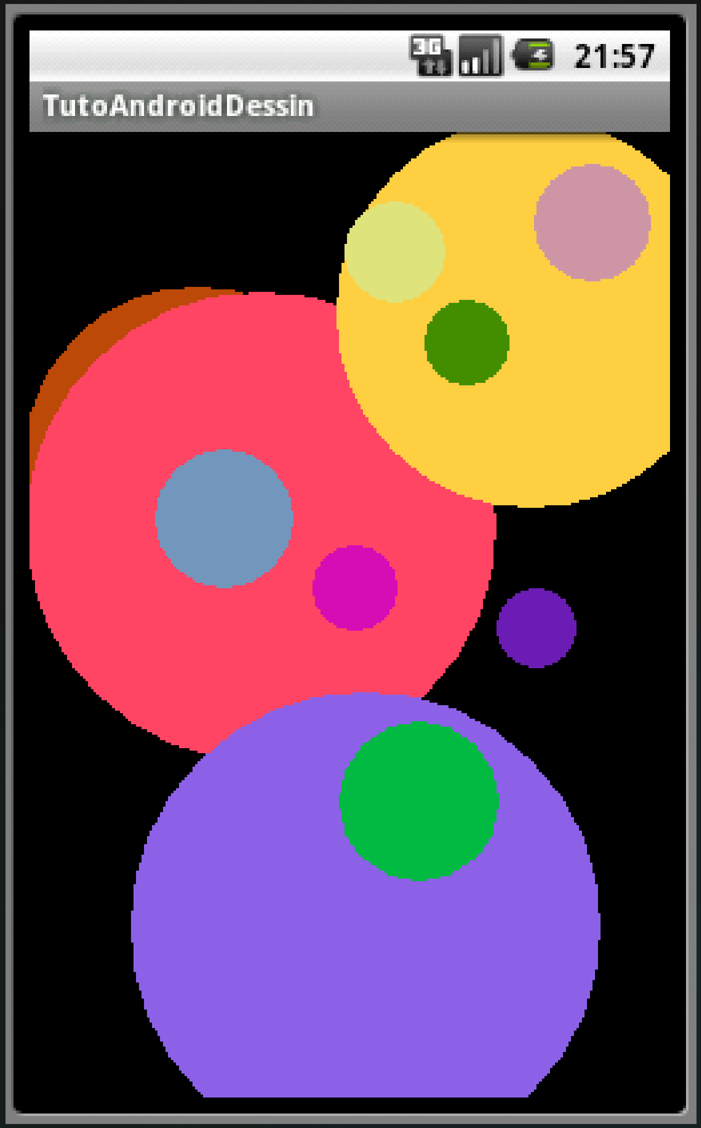
- Méthode insert = wrapper pour un INSERT

Mise à jour des données

- Au retour de l'activité de saisie, dans l'activité principale, il faut mettre à jour la liste :

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    cListeInfos.requery();
}
```

Une application de dessin



Pour finir par quelque chose de joli...

Dessiner sur Android

- En 3D : OpenGL, pas pour aujourd'hui
- En 2D :
  - Faire une sous-classe de View mise dans le layout
  - Les méthodes de cette vue perso :
    - dessinent l'image
    - gèrent les touchers de l'écran

Organisation globale du projet

- Deux classes principales :
  - Activité principale : quasi vide
    - Son layout contient une View perso qui fait TOUT !
  - View perso Dessin : gros morceau
    - sa méthode onDraw dessine tout
    - sa méthode onTouch gère les appuis du doigt
    - Une classe interne Cercle (centre , rayon)

C'est parti !

- Créer un nouveau projet Android :
  - Nom = TutoAndroidDessin
  - Package = fr.iutlan.tutodessin
  - Activité = TutoAndroidDessinActivity
- Laisser intact le code généré
  - On va seulement éditer le layout
  - Ce layout référence une nouvelle vue perso qu'on va programmer

Edition du layout main.xml

- Voici ce que devient le layout :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent" android:layout_height="fill_parent">
   <fr.iutlan.tutodessin.Dessin
      android:id="@+id/dessin"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" />
</LinearLayout>
```

Layout et custom View

- La notation <*package.classe*... /> indique qu'on utilise une vue de l'application désignée par le package
- Il faut programmer une classe appelée comme ça, héritant de View :

```
package package;
import android.view.View;
public class classe extends View {
    ...
}
```

Classe Dessin extends View

- Ajouter une nouvelle classe appelée Dessin :

```
import android.graphics.Canvas;
public class Dessin extends View {
    public Dessin(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
    @Override
    public void onDraw(Canvas canvas) {
    }
}
```

Les Canvas

- Un Canvas est une zone de dessin 2D rectangulaire
- Méthodes de dessin :
  - drawPoint, drawLine, drawPath, drawRect...
  - Principes : drawMachin(coordonnées, paint)

    http://developer.android.com/reference/android/graphics/Canvas.html
- Paint = peinture à employer
  - paint.setColor, setAntiAlias...

    http://developer.android.com/reference/android/graphics/Paint.html

Méthode de dessin

- Juste pour voir, mettre ces lignes dans la méthode onDraw :

```
Paint paint = new Paint();
paint.setColor(Color.BLUE);
canvas.drawCircle(100, 100, 50, paint);
```

  - On crée une peinture bleue et on dessine un cercle
- On va améliorer cette idée :
  - dessiner plusieurs cercles
  - les créer à la souris

Classe Cercle

- Dans Dessin.java, rajouter la classe Cercle :

```
class Cercle {
    int xc, yc, rayon;
    private Paint paint;
    public Cercle(int x, int y, int r) {
        xc = x;   yc = y; rayon = r;
        paint = new Paint();
        paint.setColor(Color.rgb(RND, RND, RND));
    }
```

  - Remplacer les 3 *RND* par (int)(Math.random()*256)

Méthode de dessin pour un cercle

- Rajouter la méthode :

```
public void draw(Canvas canvas)
{
    canvas.drawCircle(xc, yc, rayon, paint);
}
```

- Donc, en faisant cercle.draw(moncanvas) le cercle se dessine sur ce canvas

Liste de cercles

- On va gérer une multitude de cercles dans la classe Dessin :

```
import java.util.LinkedList;
LinkedList<Cercle> cercles;
public Dessin(Context context, AttributeSet attrs) {
    super(context, attrs);
    cercles = new LinkedList<Cercle>() ;
}
```

Méthode onDraw

- La méthode de dessin devient :

```
@Override
public void onDraw(Canvas canvas) {
    for (Cercle cercle: cercles)
        cercle.draw(canvas);
}
```

- On appelle la méthode draw de chaque cercle
- Il reste à pouvoir créer les cercles avec le doigt

Réaction aux touchers écran

- Pour qu'une vue réagisse aux touchers, elle doit implémenter l'interface OnTouchListener
  - Elle doit simplement posséder une méthode **onTouch**
- **onTouch**(View v, MotionEvent **event**) est appelée dans au moins deux cas :
  - event.getAction() == ACTION_DOWN : on pose le doigt en event.getX(), event.getY()
  - event.getAction() == ACTION_MOVE : on bouge le doigt en event.getX(), event.getY()

Réaction aux touchers écran

- Voici ce qu'il faut ajouter à la classe Dessin :

```
import android.view.View.OnTouchListener;
public class Dessin extends View implements OnTouchListener {
  @Override
  public boolean onTouch(View v, MotionEvent event) {
    int x = (int)event.getX(); int y = (int)event.getY();
    switch (event.getAction()) {
      case MotionEvent.ACTION_DOWN: // toucher en x,y
      case MotionEvent.ACTION_MOVE:  // mouvement vers x,y
    }
  }
```

On arrive presque au bout

- Cas ACTION_DOWN quand on clique, on rajoute un nouveau cercle en x,y :

```
cercles.add(new Cercle(x, y, 1));
```

- Cas ACTION_MOVE quand on bouge le doigt, on change le rayon du dernier cercle créé :

```
Cercle cercle = cercles.getLast();
cercle.rayon = (int)Math.sqrt(
    (x-cercle.xc)*(x-cercle.xc) + (y-cercle.yc)*(y-cercle.yc));
```

- Ne pas oublier les break; dans le switch !

Mettre à jour le dessin

- Enfin, comme on a rajouté un cercle, il faut redessiner la vue :
  - Dans public boolean onTouch(...) rajouter en fin :

    ```
    this.invalidate();
    ```

  - Cette instruction signale à la vue que son contenu n'est plus à jour, elle va donc rappeler automatiquement onDraw dès que possible.

Ca y est, c'est fini !

- Des questions ?

